INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A truncated SQP algorithm for solving nonconvex equality constrained optimization problems

Laurent CHAUVIER — Antonio FUDULI — Jean Charles GILBERT

N° 4346

29 décembre 2001

THÈME 4

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4346--FR+ENG

A truncated SQP algorithm for solving nonconvex equality constrained optimization problems

Laurent CHAUVIER* , Antonio FUDULI† , Jean Charles GILBERT‡

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Estime

Rapport de recherche n° 4346 — 29 décembre 2001 — 28 pages

Abstract: An algorithm for solving equality constrained optimization problems is proposed. It can deal with nonconvex functions and uses a truncated conjugate algorithm for detecting nonconvexity. The algorithm ensures convergence from remote starting point by using line-search. Numerical experiments are reported, comparing the approach with the one implemented in the trust region codes ETR and Knitro.

Key-words: Equality constraint – exact penalty function – global convergence – line-search – Newton's method – nonconvex optimization – sequential quadratic programming – truncated conjugate gradient algorithm.

* Artelys, 215 rue Jean-Jacques Rousseau, 92136 Issy-les-Moulineaux Cedex (France); e-mail: `Laurent.Chauvier@artelys.fr`

† Dipartimento di Ingegneria dell'Innovazione, Universitá di Lecce, Via Monteroni, 73100 Lecce (Italy); e-mail: `Antonio.Fuduli@unile.@ it`

‡ INRIA Rocquencourt, projet Estime, BP 105, 78153 Le Chesnay Cedex (France); e-mail: `Jean-Charles.Gilbert@inria.fr`.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Un algorithme PQS tronqué pour résoudre les problèmes d'optimisation non convexes sous contraintes d'égalité

Résumé : Nous proposons un algorithme pour résoudre les problèmes d'optimisation sous contraintes d'égalité généraux. Il utilise le gradient conjugué tronqué pour prendre en compte la non-convexité éventuelle des fonctions définissant le problème. La recherche linéaire permet d'assurer la convergence des itérés, même en cas de démarrage en un point éloigné d'une solution. Des tests ont été réalisés sur banc d'essai (collection CUTE) et sur des problèmes d'optique ophtalmique. Les résultats numériques comparent l'approche proposée à celle implémentée dans les codes à régions de confiance ETR et Knitro.

Mots-clés : Contraintes d'égalité – convergence globale – fonction de pénalisation exacte – gradient conjugué tronqué – méthode de Newton – optimisation non convexe – programmation quadratique successive – recherche linéaire.

1 Introduction

In this paper, we consider with an algorithmic viewpoint the problem of minimizing a nonlinear function on a nonlinear manifold defined by equality constraints. The problem is written as follows

$$\begin{array}{rl} \text{minimize} & f(x) \\ \text{subject to} & c(x) = 0, \end{array} \tag{1.1}$$

where the objective $f : \mathbb{R}^n \to \mathbb{R}$ and the constraints $c : \mathbb{R}^n \to \mathbb{R}^m$ are smooth functions. Our approach accepts nonconvex functions, but requires f and c to be twice differentiable. Hessian-vector products are indeed used to speed up the convergence and to deal with the possible nonconvexity of the problem. In practice, these products could be approximated by finite differences of gradients, but we do not report here any experiment along this line.

The algorithm uses line-search and can be viewed as a natural extension of the truncated Newton method of Dembo and Steihaug [11] for solving unconstrained minimization problems. Despite its simplicity, we have not found any published description of such an algorithm, although it is linked to the Steihaug technique for solving approximately the quadratic subproblems in the trust region approach (see [28, 25, 10]). Several details of the proposed algorithm differ from those used in the trust region framework, because of the need to generate descent directions. The extension of this algorithm to problems with inequality constraints with a nonlinear interior point approach is considered in [8] and the forthcoming paper [14].

The trust region (TR) approach is often presented as a robust technique to solve nonconvex constrained optimization problems and numerical experiments have confirmed this viewpoint (see the impressive [10] for a state of the art on TRs). However, this approach still needs modifications to solve efficiently large scale optimal control problems (OCPs). According to us, their main current limitations may be due to the use of spherical trust regions. First, this makes preconditioning difficult. Second, spherical trust regions have the propensity to force the use of a restoration operator that is perpendicular to the space tangent to the constraint manifold (the restoration step in [5, 26, 18] has this property when the trust region does not intersect the linearized constraint manifold). This aspect of the algorithm is time consuming in many large scale OCPs. On the other hand, the proposed line-search algorithm is probably less robust than those using trust region techniques (for example, the method cannot deal with pointwise singular constraint Jacobians and it is not guaranteed that its limit points are at least stationary points of the norm of the constraints), but it encounters no particular difficulties with preconditioning and nonnormal restoration operators. We believe also that it can be very helpful for solving a large class of OCPs, because of its ability to take advantage of their structure, as we now explain.

From the optimization viewpoint, a discretized OCP is a general nonlinear problem like (1.1), with the following structure (inequality constraints are often present in these problems, but we shall not consider that general case here; see [1] for a recent overview on the use of mathematical programming techniques to solve OCPs). The variables $x = (x_1, \dots, x_n)$ to optimize are partitioned in $x = (y, u)$. The components of $y \in \mathbb{R}^m$ (as many as equality constraints), called *state variables*, describe the state of the system under study, while the components of $u \in \mathbb{R}^{n-m}$, called *control variables*, are parameters that can be used to modify and control the state of the system. Accordingly, the $m \times n$ Jacobian matrix of the equality constraints $A(x) := c'(x)$ is partitioned as follows

$$A(x) = \begin{pmatrix} B & N \end{pmatrix},$$

where $B = B(x) = \frac{\partial c}{\partial y}(x)$ and $N = N(x) = \frac{\partial c}{\partial u}(x)$. A key feature of OCPs is that it is reasonable to assume the nonsingularity of the matrix B. In this case, by the implicit function theorem, the state variable y can be expressed as a function of the control variable u and the equality constraint $c(y, u) = 0$, also called *state equation*, reflects how the state y varies when u is changed.

Often, engineers have already an extensive experience in modelization when they come to optimization. The system they study is described by some equation, $F(y) = 0$ say, with $F : \mathbb{R}^m \to \mathbb{R}^m$. The question then arises to know how to optimize the system, with respect to some criterion, by modifying some parameters u. The model equation is then written $c(y, u) = 0$, the previous equation being recovered for some value u_0 of the control parameter: $F(\cdot) \equiv c(\cdot, u_0)$. A typical example is shape optimization in hydrodynamics, in which the boundary of the domain of interest (described by shape parameters) has to be designed to obtain optimal properties of the simulated flow [21]. Another example is the determination of optimal trajectories in the presence of obstacles (see [9, 8] for a problem where a deep tethered vehicle is controlled by a towing ship; this problem has been used as test-problem in the design of our algorithm).

These applications have in common that the state of the system is computed by solving the equation $F(y) = 0$ by (possibly damped) Newton's iterations. Specific research may have yielded efficient techniques for solving the linear system $B\bar{r} = -F$ defining the Newton step $\bar{r}$. We think of exploiting the sparsity of the matrix $B = F'$ or using parallelism, for instance. The optimization algorithm that we propose tries to use as much as possible the fact that, for OCPs, the step $\bar{r}$ is a good displacement for computing the state of the system. From this viewpoint, our optimization algorithm is a technique modifying the direction $\bar{r}$ in order to reach optimality. This has also the advantage of allowing the algorithm to use adapted numerical techniques developed before the optimization has come into play.

The paper is organized as follows. Section 2 presents the properties that are helpful to design the algorithm. In section 3, we describe the algorithm and show its global convergence. Finally, section 4 relates some numerical experiments, comparing our approach with the TR codes ETR and Knitro.

Notation

We denote by $\|\cdot\|$ the ℓ_2 or Euclidean norm.

2 The search direction

Let us denote by $A(x)$ the $m \times n$ Jacobian matrix of c at x. It will always be assumed to have full row rank. Then, for any solution x_* of (1.1) there exists a unique Lagrange multiplier $\lambda_* \in \mathbb{R}^m$ such that (see for example [13]):

$$\left\{ \begin{array}{l} \nabla f(x_*) + A(x_*)^\top \lambda_* = 0 \\ c(x_*) = 0. \end{array} \right. \tag{2.2}$$

The first equation is the gradient (associated with the Euclidean scalar product) with respect to x of the *Lagrangian* function

$$(x, \lambda) \in \mathbb{R}^n \times \mathbb{R}^m \mapsto \ell(x, \lambda) := f(x) + \lambda^\top c(x) \in \mathbb{R}.$$

Its Hessian with respect to x is denoted by

$$L(x, \lambda) := \nabla^2_{xx} \ell(x, \lambda).$$

A point x_* satisfying (2.2) for some λ_* is called a *stationary point*. At a solution (x_*, λ_*) of (1.1), $L_* := L(x_*, \lambda_*)$ is positive semi-definite on $N(A_*)$, the null space of $A_* := A(x_*)$. We say that (x_*, λ_*) is a *strong solution* if L_* is positive definite on $N(A_*)$: $d^\top L_* d > 0$ for all nonzero $d \in N(A_*)$. A stationary point x_*, with associated multiplier λ_*, is said to be *regular* if A_* is surjective and if any $d \in N(A_*)$ such that $L_* d \in N(A_*)^\perp$ vanishes. When the constraint Jacobian is surjective, a strong solution is an example of regular stationary point (see [4]).

The standard version of the SQP algorithm for solving (1.1) is a Newton-like method for finding a solution of (2.2) (see for example [13, 2, 27, 25, 4]). An iteration starting at (x, λ) first solves the following linear system for $(d, \lambda^{\mathrm{QP}})$:

$$\begin{pmatrix} M & A(x)^\top \\ A(x) & 0 \end{pmatrix} \begin{pmatrix} d \\ \lambda^{\mathrm{QP}} \end{pmatrix} = - \begin{pmatrix} \nabla f(x) \\ c(x) \end{pmatrix} \tag{2.3}$$

where M is a symmetric matrix, which can be indefinite. In Newton's method M is the Hessian of the Lagrangian $L(x, \lambda)$ and in quasi-Newton methods M is updated at each iteration to approximate $L(x, \lambda)$. In these cases, M depends directly or indirectly on λ. Next, (x, λ) is updated by

$$x_+ = x + d \quad \text{and} \quad \lambda_+ = \lambda^{\mathrm{QP}}.$$

The pair $(d, \lambda^{\mathrm{QP}})$ is also a primal-dual stationary point of the quadratic problem

$$\begin{array}{ll} \min & \nabla f(x)^\top d + \dfrac{1}{2} d^\top M d \\ \text{s.t.} & A(x) d + c(x) = 0. \end{array} \tag{2.4}$$

It is known that, if the first iterate (x_0, λ_0) is close enough to a regular stationary pair (x_*, λ_*) of (1.1), the SQP algorithm with $M = L(x, \lambda)$ is well defined (i.e., (2.3) has a unique solution) and generates a sequence $\{(x_k, \lambda_k)\}_{k \geq 0}$ of primal-dual pairs converging quadratically to (x_*, λ_*).

2.1 System reduction

To compute a solution of the linear system (2.3), one can proceed as follows. The dual solution λ^{QP} being determined by the value of d and the first equation of (2.3) (remember that we assume that $A(x)$ is surjective), the main task consists in determining d.

Any direction d satisfying the linearized constraints can be written $d = r + t$, where r is a particular solution of

$$A(x)r = -c(x)$$

and t is a displacement in $N(A)$, the null space of $A(x)$. The displacement r is called the *restoration step* and t is called the *tangent step* since it is tangent to the manifold $c^{-1}(c(x))$. There are several meaningful ways of decomposing d in a restoration step and tangent step.

As we said in the introduction, in OCPs, x is partitioned in (y, u) and the Jacobian matrix $A(x)$ is similarly partitioned in

$$A(x) = \begin{pmatrix} B(x) & N(x) \end{pmatrix} = \begin{pmatrix} \frac{\partial c}{\partial y}(y,u) & \frac{\partial c}{\partial u}(y,u) \end{pmatrix},$$

with $B(x)$ nonsingular. The vector r can then be computed by

$$r = -\begin{pmatrix} B(x)^{-1} \\ 0 \end{pmatrix} c(x). \tag{2.5}$$

Clearly, the first m components of r, $-B(x)^{-1}c(x)$, is the Newton step to solve the state equation $c(\cdot, u) = 0$ with fixed control parameters u. In OCPs, computing r as above is often the most straightforward and natural approach. For example, when the constraint comes from the discretization of a differential equation, B is a sparse lower block triangular matrix. It is therefore very attractive to compute r in that way and to have an optimization algorithm where this step is allowed. On the other hand, the columns of

$$\begin{pmatrix} -B(x)^{-1}N(x) \\ I \end{pmatrix} \tag{2.6}$$

form a basis of $N(A(x))$, so that the tangent step can be taken in the range space of this matrix.

More generally, since $A(x)$ is assumed to have full row rank, it has a right inverse: this is an injective matrix $A^{-}(x) \in \mathbb{R}^{n\times m}$ such that

$$A(x)A^{-}(x) = I_m. \tag{2.7}$$

The matrix factor of $-c(x)$ in (2.5) is an example of right inverse of $A(x)$, which is adapted to OCPs. The restoration step can then be computed by $r = -A^{-}(x)c(x)$. The algorithm does not require that the matrix $A^{-}(x)$, nor its transpose, be explicitly formed. Only products of these matrices with various vectors are needed. Let also

$Z^-(x)$ be an $n \times (n-m)$ matrix whose columns form a basis of $N(A(x))$, i.e., $Z^-(x)$ is injective and

$$A(x)Z^-(x) = O_{m\times(n-m)}. \tag{2.8}$$

The *reduced gradient* associated with the "basis" $Z^-(x)$ is the vector of $\mathbb{R}^{n-m}$ defined by

$$g(x) := Z^-(x)^\top \nabla f(x). \tag{2.9}$$

In OCPs, the matrix (2.6) is a natural basis and the reduced gradient is often computed by first solving the so-called *adjoint equation*

$$B(x)^\top p = \nabla_y f(x)$$

and next $g(x) = \nabla_u f(x) - N(x)^\top p$.

In the sequel, we suppose that for any x the matrices $A^-(x)$ and $Z^-(x)$ are given and adapted to the problem to solve. We want to design an optimization algorithm that uses these matrices, without modifying them by costly computations.

In the formalism given above, any solution of the linearized constraints can be written

$$d = r + t$$

with

$$r = -A^-(x)c(x) \quad \text{and} \quad t = Z^-(x)u,$$

for some $u \in \mathbb{R}^{n-m}$ to determine. This is a first way of decomposing d. Substituting this expression of d into the first equation of (2.3) and multiplying on the left by $Z^-(x)^\top$ yields the so-called *reduced system*

$$Hu = v, \tag{2.10}$$

where

$$H := Z^-(x)^\top M Z^-(x) \quad \text{and} \quad v := -g(x) + Z^-(x)^\top M A^-(x)c(x). \tag{2.11}$$

The symmetric matrix H, called the *reduced matrix*, need not be assembled in our algorithm. When $M = L(x,\lambda)$, H depends on x and λ and is positive definite (resp. nonsingular) at a strong solution (resp. a regular stationary point) of (1.1).

2.2 Conjugate gradient iterations

When $M = L(x,\lambda)$, it is instructive to compare the linear system (2.10) with the Newton system in unconstrained optimization, i.e., $\nabla^2 f(x)d = -\nabla f(x)$. In both cases, the matrix of the system is symmetric and positive semi-definite at a minimum point. For unconstrained problems, it is rarely appropriate to compute an exact solution of this linear system when x is far from a minimization point: it is unlikely to be a descent direction of f (except if f is strongly convex) and it requires therefore unnecessary computational effort. In the truncated Newton approach [11], this linear system is solved more and more accurately as the iterates progress to the solution, by a controlled number of conjugate gradient (CG) iterations. By adapting the accuracy with which system is solved one can get a superlinear or quadratic rate of convergence. An interesting property of this approach is that, provided the CG is interrupted before encountering a negative curvature direction (see below), the approximate solution is a descent direction of f. We follow the same idea and solve the reduced system (2.10) inexactly by truncated CG iterations. In section 2.3, we show that this strategy provides a direction d along which some classical exact penalty merit function decreases.

The truncated conjugate gradient (TCG) algorithm for solving (2.10) approximately is presented below. Its iterations are called *inner iterations* as opposed to the *outer iterations* of the SQP algorithm. The algorithm starts with $\mathsf{u}^0 = 0$ as the initial approximation of u, generates approximate solutions u^j for $j = 0, \ldots, i$, as well as conjugate directions v^j. The negative residual is denoted by $\mathsf{r}^j = v - H\mathsf{u}^j$.

The algorithm can be stopped at any iteration, but it must certainly be interrupted at u^j if the next conjugate direction v^j is a *quasi-negative curvature* direction. We mean by this that the following inequality does not hold:

$$(\mathsf{v}^j)^\top H \mathsf{v}^j \geq \nu \|\mathsf{v}^j\|^2. \tag{2.12}$$

The parameter $\nu > 0$ is maintained constant during the CG iterations (but it will vary along the outer iterations, see section 3). Algorithm TCG will simply discard quasi-negative directions, hence also negative curvature directions (for which the left-hand side in (2.12) is negative).

ALGORITHM TCG (Truncated Conjugate Gradient):

1. Set $\mathsf{u}^0 = 0$ and $\mathsf{r}^0 = v$.
2. If $v = 0$, set $i = 0$ and go to Step 4.
3. For $j = 0, 1, \ldots$ do the following:
   3.1. Stop to iterate and go to Step 4 with $i = j$ if desired.
   3.2. Compute the jth conjugate direction:

$$\mathsf{v}^j := \begin{cases} \mathsf{r}^0 & \text{if } j = 0 \\ \mathsf{r}^j + \frac{\|\mathsf{r}^j\|^2}{\|\mathsf{r}^{j-1}\|^2}\mathsf{v}^{j-1} & \text{if } j \geq 1. \end{cases}$$

   3.3. Compute $\mathsf{p}^j = H\mathsf{v}^j$.
   3.4. If (2.12) does not hold, then go to Step 4 with $i = j$.
   3.5. Compute the stepsize:

$$\mathsf{t}^j = \frac{\|\mathsf{r}^j\|^2}{(\mathsf{v}^j)^\top \mathsf{p}^j}.$$

3.6. Compute the new iterate $\mathsf{u}^{j+1} = \mathsf{u}^j + \mathsf{t}^j \mathsf{v}^j$ and the new negative residual $\mathsf{r}^{j+1} = \mathsf{r}^j - \mathsf{t}^j \mathsf{p}^j$.

4. If $i = 0$ take $u = v$, else take $u = \mathsf{u}^i$, as approximate solution of (2.10).

It is important to note that the algorithm uses quasi-negative directions in a different way when $i = 0$ or $i \geq 1$. In the first case, u is set to that quasi-negative direction $\mathsf{v}^0 = \mathsf{r}^0 = v$, while in the latter case the quasi-negative direction v^j is discarded. In this way, the approximate solution u of (2.10) computed by the algorithm is zero only when $v = 0$.

The next proposition shows that the approximate solution $u = \mathsf{u}^i$ can be written in a compact form $\mathsf{u}^i = Jv$, where J is a positive semi-definite matrix, which can then be viewed as an approximation of the inverse of H (provided this one exists).

Proposition 2.1 *The approximate solution u of* (2.10) *computed by Algorithm TCG can be written*

$$u = Jv,$$

where J is the identity matrix if $i = 0$ and

$$J := \sum_{j=0}^{i-1} \frac{\mathsf{v}^j (\mathsf{v}^j)^\top}{(\mathsf{v}^j)^\top H \mathsf{v}^j}, \quad \textit{if } i \geq 1. \tag{2.13}$$

Proof. If $i = 0$, $u = v$ and the result follows. Otherwise Algorithm TCG generates conjugate directions $\mathsf{v}^0, \ldots, \mathsf{v}^{i-1}$, so that for $0 \leq j \leq i-1$:

$$\|\mathsf{r}^j\|^2 = (\mathsf{v}^j)^\top \mathsf{r}^j = -(\mathsf{v}^j)^\top (H\mathsf{u}^j - v) = -(\mathsf{v}^j)^\top H \left(\sum_{l=0}^{j-1} \mathsf{t}^l \mathsf{v}^l \right) + (\mathsf{v}^j)^\top v = (\mathsf{v}^j)^\top v,$$

by conjugacy of v^j and v^l. Therefore

$$u = \sum_{j=0}^{i-1} \mathsf{t}^j \mathsf{v}^j = \sum_{j=0}^{i-1} \frac{(\mathsf{v}^j)^\top v}{(\mathsf{v}^j)^\top H \mathsf{v}^j} \mathsf{v}^j = \left(\sum_{j=0}^{i-1} \frac{\mathsf{v}^j (\mathsf{v}^j)^\top}{(\mathsf{v}^j)^\top H \mathsf{v}^j} \right) v.$$

□

2.3 Descent direction

The direction produced by our algorithm adds the restoration step $r = -A^- c$ and the tangent step $t = Z^- u$, where $u = Jv$ is the approximate solution of the reduced system (2.10) computed by Algorithm TCG:

$$d = r + t = -A^- c + Z^- Jv, \tag{2.14}$$

where v is given by (2.11). Introducing the following right inverse $\widetilde{A}^- := \widetilde{A}^-(x)$ of $A := A(x)$ (use (2.7) and (2.8) to see that $A\widetilde{A}^- = I_m$):

$$\widetilde{A}^- = (I - Z^- J Z^{-\top} M) A^-, \tag{2.15}$$

the direction d can be rewritten

$$d = \widetilde{r} + \widetilde{t}, \tag{2.16}$$

where

$$\widetilde{r} = -\widetilde{A}^- c \quad \text{and} \quad \widetilde{t} = -Z^- J g. \tag{2.17}$$

Formula (2.16) gives a second way of decomposing d in its restoration and tangent steps.

Consider the merit function

$$\Theta_\sigma(x) = f(x) + \sigma \|c(x)\|_P, \tag{2.18}$$

where $\|\cdot\|_P$ is an arbitrary norm on $\mathbb{R}^m$ and σ is a positive penalty parameter. We denote by $\|\cdot\|_D$ the dual norm of $\|\cdot\|_P$, which is defined by

$$\|v\|_D = \sup_{\|u\|_P \le 1} u^\top v.$$

Note that $|u^\top v| \le \|u\|_P \|v\|_D$. It is known that if (x_*, λ_*) is a strong solution of (1.1) and if $\sigma > \|\lambda_*\|_D$, then x_* is a strict local minimum of Θ_σ (this is known as the exactness property of the penalization by Θ_σ, see [25, 4] for example). To force convergence of an algorithm using d as a basic step, it is standard to carry out a line-search at x along the direction d, forcing the decrease of Θ_σ. The parameter σ will be adapted at some iteration to ensure the exactness of Θ_σ.

The question to know whether d is a descent direction of Θ_σ at x is examined in the next proposition. The expression of the directional derivative $\Theta'_\sigma(x; d)$ makes use of the reduced gradient g defined by (2.9) and the multiplier estimate $\widetilde{\lambda}$, associated with the right inverse $\widetilde{A}^-$:

$$\widetilde{\lambda} := -\widetilde{A}^-(x)^\top \nabla f(x). \tag{2.19}$$

Proposition 2.2 *Suppose that f and c are differentiable at x. Then Θ_σ has a directional derivative at x. Its value in the direction d given by* (2.16) *is*

$$\Theta'_\sigma(x; d) = -g^\top J g + \widetilde{\lambda}^\top c - \sigma \|c\|_P. \tag{2.20}$$

It is negative if x is nonstationary and $\sigma > \|\widetilde{\lambda}\|_D$.

PROOF. Since a norm is Lipschitz continuous and has directional derivatives, $\|\cdot\|_P \circ c$ has directional derivatives and the chain rule applies: $(\|\cdot\|_P \circ c)'(x; d) = (\|\cdot\|_P)'(c; Ad) =$

$(\|\cdot\|_P)'(c;-c) = -\|c\|_P$. The last equalities come from the fact that d satisfies the linearized constraints and from the very definition of a directional derivative. Therefore

$$\Theta'_\sigma(x;d) = \nabla f^\top d - \sigma\|c\|_P.$$

Using (2.16) and (2.19), we get (2.20).

Suppose now that $\sigma > \|\widetilde{\lambda}\|_D$. Using $\widetilde{\lambda}^\top c \le \|\widetilde{\lambda}\|_D\|c\|_P$, we obtain

$$\Theta'_\sigma(x;d) \le -g^\top Jg + (\|\widetilde{\lambda}\|_D - \sigma)\|c\|_P \le 0.$$

If $\Theta'_\sigma(x;d) = 0$, it follows that $c = 0$ and $g^\top Jg = 0$. If $i = 0$ is set by Algorithm TCG, $J = I$ and therefore $g = 0$. Now i cannot be ≥ 1, since otherwise one would have $v \ne 0$ (see Step 2 of Algorithm TCG) and therefore $g \ne 0$ (since $c = 0$). But with the structure of J and the fact that $\mathsf{v}^0 = v = -g$ when $c = 0$, one would have $g^\top Jg \ge (g^\top v)^2/(v^\top Hv) = \|g\|^4/(g^\top Hg)$, which would contradict the fact that $g^\top Jg = 0$. Hence x is stationary. $\square$

Proposition 2.2 shows that if σ is larger than the computable threshold $\|\widetilde{\lambda}\|_D$, the direction d, whose tangent component is determined by Algorithm TCG, is a descent direction of Θ_σ. We use this fact in section 3 to design a line-search algorithm, in which Θ_σ is decreased at each iteration. Before this, let us show how $\widetilde{\lambda}$ can be computed inexpensively.

2.4 Computation of $\widetilde{\lambda}$

According to formulas (2.19) and (2.15), the definition of $\widetilde{\lambda}$ involves the matrix J. This matrix is formed with the conjugate directions v^j generated by Algorithm TCG; see (2.13). We do not want to store these vectors or the matrix J, however, since this would be in opposition with the low memory requirement of the CG algorithm. In fact, a closer look at the definition of $\widetilde{\lambda}$ shows that it is sufficient to evaluate $-Jg$. This vector is an approximate solution of the linear system

$$H\widetilde{u} = -g, \tag{2.21}$$

obtained by using the same conjugate directions v^j and the same Hessian-vector products $\mathsf{p}^j = H\mathsf{v}^j$, $j = 0, \ldots, i-1$, as those used to compute u as an approximate solution of (2.10). This claim will be easy to verify in a moment.

Algorithm TCG2 below computes this approximate solution $\widetilde{u}$ of (2.21) by using the vectors v^j and p^j in sequence, so that the computation can be made in parallel with the one of u, without having to store these vectors. This algorithm also needs to update the approximate solution $\tilde{\mathsf{u}}^j$ and the negative residual $\tilde{\mathsf{r}}^j = -(H\tilde{\mathsf{u}}^j + g)$ associated with (2.21).

ALGORITHM TCG2:

1. Set $\mathsf{u}^0 = 0$ and $\mathsf{r}^0 = v$. Set $\tilde{\mathsf{u}}^0 = 0$ and $\tilde{\mathsf{r}}^0 = -g$.

2. If $v = 0$, set $i = 0$ and go to Step 4.
3. For $j = 0, 1, \ldots$ do the following:
   3.1. Stop to iterate and go to Step 4 with $i = j$ if desired.
   3.2. Compute the jth conjugate direction:

$$\mathsf{v}^j := \begin{cases} \mathsf{r}^0 & \text{if } j = 0 \\ \mathsf{r}^j + \frac{\|\mathsf{r}^j\|^2}{\|\mathsf{r}^{j-1}\|^2}\mathsf{v}^{j-1} & \text{if } j \geq 1. \end{cases}$$

   3.3. Compute $\mathsf{p}^j = H\mathsf{v}^j$.
   3.4. If (2.12) does not hold, then go to Step 4 with $i = j$
   3.5. Compute the stepsizes:

$$\mathsf{t}^j = \frac{\|\mathsf{r}^j\|^2}{(\mathsf{v}^j)^\top \mathsf{p}^j} \quad \text{and} \quad \tilde{\mathsf{t}}^j = \frac{(\tilde{\mathsf{r}}^j)^\top \mathsf{v}^j}{(\mathsf{v}^j)^\top \mathsf{p}^j}.$$

   3.6. Compute the new iterates

$$\mathsf{u}^{j+1} = \mathsf{u}^j + \mathsf{t}^j\mathsf{v}^j \quad \text{and} \quad \tilde{\mathsf{u}}^{j+1} = \tilde{\mathsf{u}}^j + \tilde{\mathsf{t}}^j\mathsf{v}^j$$

   and the new negative residuals

$$\mathsf{r}^{j+1} = \mathsf{r}^j - \mathsf{t}^j\mathsf{p}^j \quad \text{and} \quad \tilde{\mathsf{r}}^{j+1} = \tilde{\mathsf{r}}^j - \tilde{\mathsf{t}}^j\mathsf{p}^j.$$

4. If $i = 0$ take $u = v$, else take $u = \mathsf{u}^i$, as approximate solution of (2.10).
   If $i = 0$ take $\widetilde{u} = -g$, else take $\widetilde{u} = \tilde{\mathsf{u}}^i$, as approximate solution of (2.21).

It is not difficult to show that, as announced, the approximate solution $\widetilde{u}$ of (2.21) computed by Algorithm TCG2 can be written

$$\widetilde{u} = -Jg. \tag{2.22}$$

Indeed, if $i = 0$, Algorithm TCG2 takes $\widetilde{u} = -g$ and (2.22) follows since $J = I$. If $i \geq 1$, Algorithm TCG2 takes

$$\widetilde{u} = \tilde{\mathsf{u}}^i = \sum_{j=0}^{i-1} \tilde{\mathsf{t}}^j\mathsf{v}^j = \sum_{j=0}^{i-1} \frac{(\tilde{\mathsf{r}}^j)^\top \mathsf{v}^j}{(\mathsf{v}^j)^\top \mathsf{p}^j}\,\mathsf{v}^j.$$

Now, using an argument similar to the one in the proof of Proposition 2.1, one has

$$(\mathsf{v}^j)^\top \tilde{\mathsf{r}}^j = -(\mathsf{v}^j)^\top (H\tilde{\mathsf{u}}^j + g) = -(\mathsf{v}^j)^\top H \left(\sum_{l=0}^{j-1} \tilde{\mathsf{t}}^l \mathsf{v}^l \right) - (\mathsf{v}^j)^\top g = -(\mathsf{v}^j)^\top g,$$

by conjugacy of v^j and v^l. Therefore

$$\widetilde{u} = -\left(\sum_{j=0}^{i-1} \frac{\mathsf{v}^j(\mathsf{v}^j)^\top}{(\mathsf{v}^j)^\top H \mathsf{v}^j} \right) g = -Jg$$

and (2.22) follows again.

Finally, the multiplier that is used in the algorithm for setting the lower threshold $\|\widetilde{\lambda}\|_D$ for the penalty parameter σ (see section 2.3) is computed by

$$\widetilde{\lambda} = -A^{-\top}(\nabla f + MZ^{-}\widetilde{u}), \tag{2.23}$$

where $\widetilde{u}$ is now the *approximate* solution (2.22) of (2.21) computed in Algorithm TCG2. Note that, when H is positive definite and $J = H^{-1}$, $\widetilde{\lambda}$ is not the QP multiplier λ^{QP}, but the multiplier of the problem

$$\begin{array}{ll} \min & \nabla f^{\top}\widetilde{t} + \frac{1}{2}\widetilde{t}^{\top}M\widetilde{t} \\ \text{s.t.} & A\widetilde{t} = 0. \end{array} \tag{2.24}$$

An approximation of λ^{QP} could also be obtained by

$$\lambda^{\mathrm{QP}} \simeq \widetilde{\lambda} - A^{-\top}MZ^{-}(u - \widetilde{u}) + A^{-\top}MA^{-}c.$$

This approximation could be useful for TR algorithms, but we shall not need it in the present context.

3 The algorithm and its global convergence

The overall algorithm for solving Problem (1.1) generates a sequence $\{x_k\}_{k\geq 0}$ by the recurrence

$$x_{k+1} = x_k + \alpha_k d_k,$$

where the direction $d_k \in \mathbb{R}^n$ is determined by (2.16)-(2.17) as in section 2 and the stepsize $\alpha_k > 0$ is determined by a line-search along d_k on the merit function Θ_{σ_k} defined by (2.18). In this section, all quantities depending on the iteration index k receive a subscript k: v_k for v, v_k^j for v^j, *etc.* We also note $c_k = c(x_k)$, $\nabla f_k = \nabla f(x_k)$, $g_k = g(x_k)$, *etc.*

The tangent part of the direction d_k depends on the number of CG iterations performed in Algorithm TCG2. In turn, this depends on the quasi-negative curvature threshold ν_k, which is allowed to vary from iteration to iteration. One would like to take ν_k small when x_k is close to a solution and larger far away. The speed of convergence depends indeed on the precision with which the reduced system (2.10) is solved close to the solution. Arbitrary small $\nu_k > 0$ can prevent convergence. The only condition required by the convergence theory below is the following:

$$\left\{ \begin{array}{l} \text{if for some sequence } \mathcal{K} \subset \mathbb{N}, \\ \text{there exists } \gamma > 0 \text{ such that, for all } k \in \mathcal{K},\ \|c_k\| + \|g_k\| \geq \gamma, \\ \text{then, there exists } \bar{\nu} > 0 \text{ such that, for all } k \in \mathcal{K},\ \nu_k \geq \bar{\nu}. \end{array} \right. \tag{3.25}$$

For example, the rule $\nu_k \geq \min(\nu', \nu''(\|c_k\| + \|g_k\|))$, where ν' and ν'' are positive constants, satisfies this assumption.

Proposition 2.2 has shown us that d_k is a descent direction of Θ_{σ_k} provided x_k is nonstationary and $\sigma_k > \|\widetilde{\lambda}_k\|_D$. To get convergence of the algorithm, however, it is necessary to ask slightly more on the penalty parameter σ_k. We fix a constant $\bar{\sigma} > 0$ and assume that

$$\left\{\begin{array}{ll} \text{(a)} & \text{for all } k,\ \sigma_k \geq \|\widetilde{\lambda}_k\|_D + \bar{\sigma}, \\ \text{(b)} & \text{there exists an index } k_0 \text{ such that:} \\ & \text{if } k \geq k_0 \text{ and } \sigma_{k-1} \geq \|\widetilde{\lambda}_k\|_D + \bar{\sigma}, \text{ then } \sigma_k = \sigma_{k-1}, \\ \text{(c)} & \text{if } \{\sigma_k\} \text{ is bounded, } \sigma_k \text{ is modified finitely often.} \end{array}\right. \tag{3.26}$$

Conditions (a)-(b) imply that after a finite number of iterations, $\{\sigma_k\}$ is nondecreasing. By condition (c), in the favorable case when $\{\sigma_k\}$ is bounded, the merit function Θ_{σ_k} is no longer modify for large iteration indices, so that it is always the same function that is decreased (this is a key point to have convergence). These properties are satisfied, for example, by the Mayne and Polak [20] update rule (the constant 1.5 is given to be specific; actually, any constant > 1 is convenient):

> **if** $\sigma_{k-1} \geq \|\widetilde{\lambda}_k\|_D + \bar{\sigma}$ **then**
> $\sigma_k = \sigma_{k-1}$
> **else**
> $\sigma_k = \max(1.5\,\sigma_{k-1}, \|\widetilde{\lambda}_k\|_D + \bar{\sigma})$
> **end if**

Assume now that σ_k satisfies (3.26). Since at a nonstationary iterate x_k, d_k is a descent direction of Θ_{σ_k}, one can determine a stepsize $\alpha_k > 0$ such that the *Armijo condition*

$$\Theta_{\sigma_k}(x_k + \alpha_k d_k) \leq \Theta_{\sigma_k}(x_k) + \omega \alpha_k \Theta'_{\sigma_k}(x_k, d_k) \tag{3.27}$$

holds. In (3.27), ω is a constant chosen in $]0, \frac{1}{2}[$. In the algorithm, α_k is determined by backtracking.

We can now summarize the overall algorithm for solving the equality constrained problem (1.1).

ALGORITHM TSQP (Truncated SQP):

1. *Initialization.* Set $k = 0$. Choose an initial iterate $(x_0, \lambda_0) \in \mathbb{R}^n \times \mathbb{R}^m$ and set the constants $\nu' > 0$, $\nu'' > 0$ (quasi-negative curvature constants), $\omega \in\ ]0, \frac{1}{2}[$ (slope modifier in the Armijo condition), $\bar{\sigma} > 0$ (penalty parameter threshold), and $\beta \in\ ]0, \frac{1}{2}]$ (backtracking safeguard parameter).
2. For $k = 0, 1, 2, \ldots$ do the following:
   2.1. *Stopping test*: Stop if $c_k = 0$ and $g_k = 0$.

2.2. *Step computation*:
- Compute the restoration step $r_k = -A_k^- c_k$.
- Run Algorithm TCG2 described in section 2.4, with a quasi-negative curvature threshold ν_k satisfying condition (3.25), to compute an approximate solution u_k of (2.10) and an approximate solution $\widetilde{u}_k$ of (2.21).
- Compute the tangent step $t_k = Z_k^- u_k$.
- Compute the total step $d_k = r_k + t_k$.

2.3. *Penalty parameter setting*:
- Compute $\widetilde{\lambda}_k$ by (2.23).
- Update σ_k such that (3.26) holds.

2.4. *Linesearch*:
- Set $\alpha = 1$.
- While α does not satisfy Armijo's inequality (3.27), take a new stepsize α in $[\beta\alpha, (1-\beta)\alpha]$.
- Set $\alpha_k = \alpha$.

2.5. *New iterates*: Set $x_{k+1} = x_k + \alpha_k d_k$ and $\lambda_{k+1} = \lambda_k^{\text{LS}}$.

Before proving the global convergence of this algorithm, let us make some observations. In a concrete implementation of this algorithm, the stopping test in Step 2.1 should be replaced by a condition checking that c_k and g_k are sufficiently small. Also in Step 2.4, the new stepsize chosen in the interval $[\beta\alpha, (1-\beta)\alpha]$ during the line-search should be determined by interpolation. In Step 2.5, we have set the new multiplier λ_{k+1} to the least-squares multiplier

$$\lambda_k^{\text{LS}} := -A_k^{-\top} \nabla f_k.$$

The interest of this choice is that this multipler estimate does not depend on second derivatives. In contrast, using $\widetilde{\lambda}_k$ or λ_k^{QP} may not be always faithful during the first iterations of the algorithm.

Here are the assumptions that are necessary to have global convergence of Algorithm TSQP. The convergence proof does not require to have $M_k = \nabla^2_{xx}\ell(x_k, \lambda_k)$, so that there is no assumptions related directly to the second derivatives of f and c. In practice, however, this is when second derivatives are used that Algorithm TSQP is the most useful.

Assumptions 3.1 (i) The functions f and c are continuously differentiable with Lipschitz continuous derivatives. (ii) The sequences $\{Z_k^-\}$, $\{A_k^-\}$, $\{M_k\}$, and $\{\widetilde{\lambda}_k\}$ generated by Algorithm TSQP are bounded.

Theorem 3.2 *Suppose that Assumptions* 3.1 *hold. Then the sequence of penalty parameters* $\{\sigma_k\}$ *is stationary for* k *sufficiently large:* $\sigma_k = \sigma$. *If furthermore* $\{\Theta_\sigma(x_k)\}$ *is bounded below, then the sequences* $\{c_k\}$ *and* $\{g_k\}$ *converge to* 0.

PROOF. We denote by C_1, C_2, ... positive constants. By (3.26)-(b) and the boundedness of $\{\widetilde{\lambda}_k\}$, $\{\sigma_k\}$ is bounded, hence stationary for k large (by (3.26)-(c)). The first part of the theorem is proved.

Since from some index, say k_0, the penalty parameters σ_k have the common value σ, the Armijo inequality (3.27) shows that $\Theta_\sigma(x_k)$ decreases. This sequence is also bounded below (by assumption), hence it converges. This implies that $\alpha_k\Theta'_\sigma(x_k;d_k)$ tends to 0, or equivalently (use Proposition 2.2 and (3.26)-(a))

$$\alpha_k g_k^\top J_k g_k \to 0 \quad \text{and} \quad \alpha_k c_k \to 0. \tag{3.28}$$

We proceed by contradiction assuming that there is an unbounded subsequence $\mathcal{K}$ of indices k and a positive constant γ such that

$$\|g_k\| + \|c_k\| \geq \gamma, \quad \text{for } k \in \mathcal{K}. \tag{3.29}$$

By condition (3.25), this implies that there is a constant $\bar{\nu} > 0$ such that $\nu_k \geq \bar{\nu}$, for $k \in \mathcal{K}$.

Let us now show that $\{\alpha_k\}_{k\in\mathcal{K}}$ is bounded away from 0. By the line-search (Step 2.4), when $\alpha_k < 1$, there is a stepsize $\bar{\alpha}_k \in \,]0,1]$ such that $\alpha_k \in [\beta\bar{\alpha}_k, (1-\beta)\bar{\alpha}_k]$ and

$$\Theta_\sigma(x_k + \bar{\alpha}_k d_k) > \Theta_\sigma(x_k) + \omega\bar{\alpha}_k\Theta'_\sigma(x_k;d_k).$$

Using the smoothness of f and c and the fact that d_k satisfies the linearized constraints, one has successively

$$\begin{aligned}
f(x_k + \bar{\alpha}_k d_k) &= f(x_k) + \bar{\alpha}_k f'(x_k)\cdot d_k + O(\bar{\alpha}_k^2\|d_k\|^2),\\
c(x_k + \bar{\alpha}_k d_k) &= (1-\bar{\alpha}_k)c(x_k) + O(\bar{\alpha}_k^2\|d_k\|^2),\\
\Theta_\sigma(x_k + \bar{\alpha}_k d_k) &\leq \Theta_\sigma(x_k) + \bar{\alpha}_k\Theta'_\sigma(x_k;d_k) + C_1\bar{\alpha}_k^2\|d_k\|^2.
\end{aligned}$$

Therefore $(\omega-1)\Theta'_\sigma(x_k;d_k) < C_1\bar{\alpha}_k\|d_k\|^2$ or

$$g_k^\top J_k g_k + \|c_k\|_P < C_2\bar{\alpha}_k\|d_k\|^2, \tag{3.30}$$

where $C_2 = C_1/((1-\omega)\min\{1,\bar{\sigma}\})$. Using $\|\mathsf{v}_k^j(\mathsf{v}_k^j)^\top\| = \|\mathsf{v}_k^j\|^2$ and $(\mathsf{v}_k^j)^\top H_k\mathsf{v}_k^j \geq \bar{\nu}\|\mathsf{v}_k^j\|^2$, one has from Proposition 2.1

$$\|J_k\| \leq \max\left(1, \sum_{j=0}^{i-1}\frac{\|\mathsf{v}_k^j(\mathsf{v}_k^j)^\top\|}{(\mathsf{v}_k^j)^\top H_k\mathsf{v}_k^j}\right) \leq \max\left(1, \frac{n-m}{\bar{\nu}}\right).$$

Hence $\{J_k\}_{k\in\mathcal{K}}$ is bounded. With the boundedness of $\{A_k^-\}$ and $\{Z_k^-\}$ (by Assumptions 3.1), the expression (2.14) of d_k yields $d_k = O(\|J_k^{1/2}v_k\| + \|c_k\|_P)$. Now, the definition (2.11) of v_k, the boundedness of $\{M_k\}$ and $\{J_k\}_{k\in\mathcal{K}}$ provide $d_k = O(\|J_k^{1/2}g_k\| + \|c_k\|_P)$. Inequality (3.30) then becomes

$$g_k^\top J_k g_k + \|c_k\|_P < C_3\bar{\alpha}_k(g_k^\top J_k g_k + \|c_k\|_P^2), \quad \text{for } k \in \mathcal{K}.$$

By (3.28), $\alpha_k c_k \to 0$ and therefore

$$g_k^\top J_k g_k < C_3 \bar{\alpha}_k g_k^\top J_k g_k, \quad \text{for } k \text{ large in } \mathcal{K}.$$

This inequality shows that $g_k^\top J_k g_k > 0$ when $\alpha_k < 1$ and k is large enough in $\mathcal{K}$ and that $\{\bar{\alpha}_k\}_{k\in\mathcal{K}}$ is bounded away from zero. Since $\alpha_k \geq \beta\bar{\alpha}_k$, $\{\alpha_k\}_{k\in\mathcal{K}}$ is also bounded away from zero. From (3.28)

$$g_k^\top J_k g_k \to 0 \quad \text{and} \quad c_k \to 0, \quad \text{for } k \in \mathcal{K}. \tag{3.31}$$

We now want to show that $g_k \to 0$ for $k \in \mathcal{K}$, which will contradict (3.29) and will conclude the proof. Observe that, with the boundedness of $\{M_k\}$ and $\{Z_k^-\}$, the formula (2.13) of J_k provides

$$g_k^\top J_k g_k \geq \frac{(g_k^\top v_k)^2}{v_k^\top H_k v_k} \geq C_4 \frac{(g_k^\top v_k)^2}{\|v_k\|^2}, \quad \text{for } k \in \mathcal{K}.$$

The numerator can be bounded below as follows: $(g_k^\top v_k)^2 = [-\|g_k\|^2 + O(\|g_k\|\,\|c_k\|)]^2 = \|g_k\|^4 + O(\|g_k\|^3\,\|c_k\|) + O(\|g_k\|^2\,\|c_k\|^2) \geq \frac{1}{2}\|g_k\|^4 - C_5\|g_k\|^2\,\|c_k\|^2$. For the denominator, we use the upper bound: $\|v_k\|^2 \leq 2\|g_k\|^2 + C_6\|c_k\|^2$. Therefore

$$g_k^\top J_k g_k \geq \frac{\frac{1}{2}\|g_k\|^2 - C_5\|c_k\|^2}{2 + C_6(\|c_k\|^2/\|g_k\|^2)}, \quad \text{for } k \in \mathcal{K}.$$

This inequality and (3.31) imply that $g_k \to 0$ for $k \in \mathcal{K}$. □

4 Numerical experiments

Algorithm TSQP has been implemented in Fortran-77, with some additional heuristics. The resulting code is denoted by `TSQP` below. In this section, we relate our experiments with this code. Actually, `TSQP` is part of a general purpose optimization software, called `Opinel`, which can also deal with inequality constraints. This latter software will be presented in a forthcoming paper [14]. The current version of the software is 0.1a.

One of the aims of these experiments is to make a comparison between the simple and flexible line-search approach implemented in `TSQP` and the robust, but more complex, trust region (TR) technique. For this reason, we have chosen to compare `TSQP` with two other TR codes: `ETR` (an equality constraint solver [18]) and `Knitro` (Version 1.00, an equality and inequality constraint solver [7, 6, 24]).

4.1 Conditions of the tests

Our benchmark is formed of a subset of test-problems from the CUTE collection [3] and some industrial-real-life test-problems provided by ESSILOR, a lens manufacturing company (see also in [17] Jonsson's contribution to this application). All the selected problems deal with equality constraint optimization. The codes have been run on a DEC-alpha PWS500 platform, under the Unix operating system.

We compare the solvers `ETR`, `Knitro`, and `TSQP`, through their *performance profiles*, a concept introduced by Dolan and Moré [12]. For the reader's convenience, we briefly summarize here the key points of this comparison principle. For a given collection of test-problems and a chosen performance criterion (such as the number of function evaluations, or gradient evaluations, *etc*), a graph is drawn with a curve relating the efficiency of each solver with respect to the other ones. For example, the first graph in figure 4.1 compares the relative performance of `ETR`, `Knitro`, `TSQP`, and `TSQP+prec` (4 curves; the code `TSQP+prec` is described below), when one considers the number of function evaluations as the performance criterion and one uses a subset of the CUTE collection as benchmark. Only three facts need to be kept in mind to have a good interpretation of these curves (for a precise definition of the curves, see the original paper [12]):

- the value given by a curve at abscissa 0 is the percentage of test-problems on which the corresponding code is the best;
- the value given by a curve at the rightmost abscissa is the percentage of test-problems that the corresponding code can solve (this is independent of the performance criterion in consideration);
- more generally, a point of a curve with coordinates $(\alpha, \phi) \in [0, +\infty[\times[0, 1]$ provides the following information: the number of test-problems for which the performance of the corresponding code is never worse than 2^α times the performance of the best code is a fraction ϕ of the total number of test-problems; to this respect, the range of values taken by α in a particular graph is meaningful.

With performance profiles, the relative efficiency of each code appears at a glance: the higher is a particular curve the better is the corresponding solver.

We have drawn these performance profiles for describing the behavior of four codes: `ETR`, `Knitro`, `TSQP`, and `TSQP+prec`. Also, four performance criteria have been selected, which leads to four graphs per benchmark: the number of function evaluations, the number of gradient evaluations, the number of Hessian-vector products $\nabla^2_{xx}\ell(x_k, \lambda_k)\, v$, and the CPU time.

In order to make the comparison meaningful, the same stopping criterion is used in all the codes. The outer iterations are stopped at the current point (x_k, λ_k) if the following conditions hold:

$$\|\nabla\ell(x_k, \lambda_k)\|_\infty \leq \varepsilon^l \tag{4.32}$$

$$\|c(x_k)\|_\infty \leq \varepsilon^c, \tag{4.33}$$

where ε^l and ε^c are positive tolerances, which may depend on the test problems and will be specified below.

4.2 Heuristics

Trust region codes like `ETR` and `Knitro` have been developed during several years. To have a chance to be competitive with them, the skeleton Algorithm TSQP needs some heuristics. We briefly describe here some of those that have been implemented in `TSQP`, to enrich the basic algorithm, and that contribute significantly to the efficiency of the solver.

Precision criterion for the CG iterations

Each CG iteration requires a Hessian-vector product $\nabla^2_{xx}\ell(x,\lambda)\,v$. This may be an expensive operation for large problems when the Hessian of the Lagrangian is not cheaply available. In order to avoid to make a large number of CG iterations at each outer iteration, the reduced system (2.10) is solved with low accuracy at the first outer iterations and progressively more precisely as the outer iteration index k increases.

`TSQP` has several ways of controlling the precision to which the linear system (2.10) has to be solved by CG iterations. To get the results presented below Nash's stopping rule [23] has been used: Algorithm TCG2 is interrupted in Step 3.1 at iteration $j \geq 1$ if

$$\frac{q_k^{j-1} - q_k^j}{-q_k^j/j} \leq \varepsilon_k^{\mathrm{CG}},$$

where

$$q_k^j = -v_k^\top \mathsf{u}^j + \frac{1}{2}\,(\mathsf{u}^j)^\top H_k \mathsf{u}^j$$

is the value at u^j of the quadratic model associated with (2.10) and $\varepsilon_k^{\mathrm{CG}} \in \,]0,1[$ is a precision threshold.

Clearly, (2.10) is solved with a higher accuracy when $\varepsilon_k^{\mathrm{CG}}$ is smaller. The number of CG iterations is then controlled by $\varepsilon_k^{\mathrm{CG}}$, which is updated by a rule using the stepsize α_{k-1} of the previous outer iteration to decide whether a higher precision is desirable at the current outer iteration. Here is the rule for updating $\varepsilon_k^{\mathrm{CG}}$.

if $k = 1$ **then**
  $\varepsilon_k^{\mathrm{CG}} = 10^{-1}$
else
  if $\alpha_{k-1} = 1$ **then**
    $\varepsilon_k^{\mathrm{CG}} = \max(10^{-7}, \frac{1}{2}\,\varepsilon_{k-1}^{\mathrm{CG}})$
  else
    $\varepsilon_k^{\mathrm{CG}} = \min(10^{-1}, 2\,\varepsilon_{k-1}^{\mathrm{CG}})$
  end if
end if

Handling negative curvature directions

`TSQP` does not implement any sophisticated technique for dealing with negative (resp. quasi-negative) curvature conjugate directions, which are those directions v_k^j for which

$(\mathsf{v}_k^j)^\top H_k \mathsf{v}_k^j$ is negative (resp. almost negative). The code just discards them by interrupting the CG iterations as in Algorithm TCG2, with a threshold ν_k (see (2.12)) that is maintained fixed to a small value.

Second order correction

Using the nondifferentiable merit function (2.18) can affect the convergence rate of Algorithm TSQP (but not its convergence) because unit stepsizes can be rejected. In order to avoid this phenomenon, known as the Maratos effect [19], we have implemented a second order correction (see for example [25, 4]).

Recall the notation for the restoration, tangent, and total steps at iteration k:

$$\widetilde{r}_k = -\widetilde{A}_k^- c(x_k), \quad \widetilde{t}_k = -Z_k^- J_k g_k, \quad \text{and} \quad d_k = \widetilde{r}_k + \widetilde{t}_k.$$

The positive constant C_{ME} below is initially set to $C_{\mathrm{ME}} := 0.1\,\|\widetilde{r}_0\|_2/\|\widetilde{t}_0\|_2$, where $\widetilde{r}_0$ and $\widetilde{t}_0$ are the initial restoration and tangent steps. It is also updated at some iterations by a rule that is not essential to specify here.

if $\Theta_{\sigma_k}(x_k + d_k) \le \Theta_{\sigma_k}(x_k) + \omega\Theta'_{\sigma_k}(x_k; d_k)$ **then**
  $x_{k+1} := x_k + d_k$
else
  if $\|\widetilde{r}_k\|_2 \le C_{\mathrm{ME}}\|\widetilde{t}_k\|_2$ **then**
    $e_k := -\widetilde{A}_k^- c(x_k + d_k)$
    if $\Theta_{\sigma_k}(x_k + d_k + e_k) \le \Theta_{\sigma_k}(x_k + d_k)$ **then**
      do an arc-search along $\alpha \mapsto x_k + \alpha d_k + \alpha^2 e_k$
    else
      do a line-search along $\alpha \mapsto x_k + \alpha d_k$
    end if
  else
    do a line-search along $\alpha \mapsto x_k + \alpha d_k$
  end if
end if

The line-search or arc-search first tries $\alpha = 1$. If this stepsize does not lead to a decrease of the merit function, the stepsize is reduced, using safeguarded interpolation. It can be shown that with this technique, the unit stepsize is accepted asymptotically.

Tangent BFGS preconditioning: `TSQP+prec`

A nice feature of `TSQP` is its ability to use "curvature" information from the reduced Hessian at previous iterations to form a preconditioning matrix for solving more rapidly the reduced systems (2.10) and (2.21) at the current iteration. This information is collected during the CG iterations, using the BFGS formula. This is quite similar to the approach proposed in [22] (see also [15] where the approach is used to accelerate

Newton's method within the TR framework and [16] for a convergence proof in the TR context). We refer to this technique as the *tangent BFGS preconditioning*.

One advantage of line-search algorithms, over the TR approach, is their ability to use tangent BFGS preconditioning, without increasing its complexity. This technique is more difficult to implement with TR, where the preconditioning is made through a modification of the trust regions. Since the updated matrix only intervenes in the tangent part of the step and provides no information on how to precondition the restoration part, tangent BFGS preconditioning suggests to modify the "tangent trust region" without affecting the region controlling the full step. The algorithm has then to control two trust regions, whose consistency is more difficult to maintain (see [17] for more details).

We have denoted by `TSQP+prec`, the version of `TSQP` that uses tangent BFGS preconditioning. Full BFGS updates are performed. We shall see that this code is very efficient on the ESSILOR problems.

4.3 Tests on the CUTE collection

In this section, we present the numerical results obtained by running `ETR`, `Knitro`, and `TSQP` on some test-problems from the CUTE collection. These have been given by the "selection facility" with the following rules: the number of variables is fixed between 1 and 100 and the number of constraints is fixed between 1 and 100. Then, 295 test-problems (from academic, modeling exercises and real life cases) have been selected. Next, we have discarded the problems with inequality constraints (the selection facility does not offer the possibility to select equality constrained problems directly) and those with $m > n$ (`argauss`, `growth`, `nystrom5`). Finally, we have not considered the problems for which the Jacobian of the constraints has not full rank at the initial point (`cluster`, `heart6`, `heart8`, `hs61`, `pfit1`, `pfit2`, `pfit3`, `pfit4` and `s316-322`), since `TSQP` is presently not designed for solving this kind of problems. As for problem `hs111lnp`, it was not possible to have the results because of a running error, appeared also when using `Knitro` and `ETR`. Although we have investigated on this, we have not been able to find a remedy. Thus it remains 60 test-problems which are given in table 4.1, together with their dimensions: n is the number of variables and m is the number of equality constraints.

The positive thresholds used in the stopping tests (4.32) and (4.33) are set in the three codes to $\varepsilon^l = \varepsilon^c = 10^{-7}$. Also, we declare a failure on a test-problem when the stopping tests cannot be satisfied in less than 1500 function evaluations.

Figure 4.1 gives the performance profiles of `ETR`, `Knitro`, `TSQP`, and `TSQP+prec`, comparing the number of function calls, gradient calls, and Hessian-vector products. We do not compare the codes on the CPU time, since this one is usually so small that its variation from run to run makes such a comparison meaningless. The selected problems have sometimes as many constraints as variables. When such is the case, the reduced space is of dimension $n - m = 0$ and there is no Hessian-vector products. As a result, the performance profiles comparing the number of Hessian-vector products were made on a subset of the selected problems.

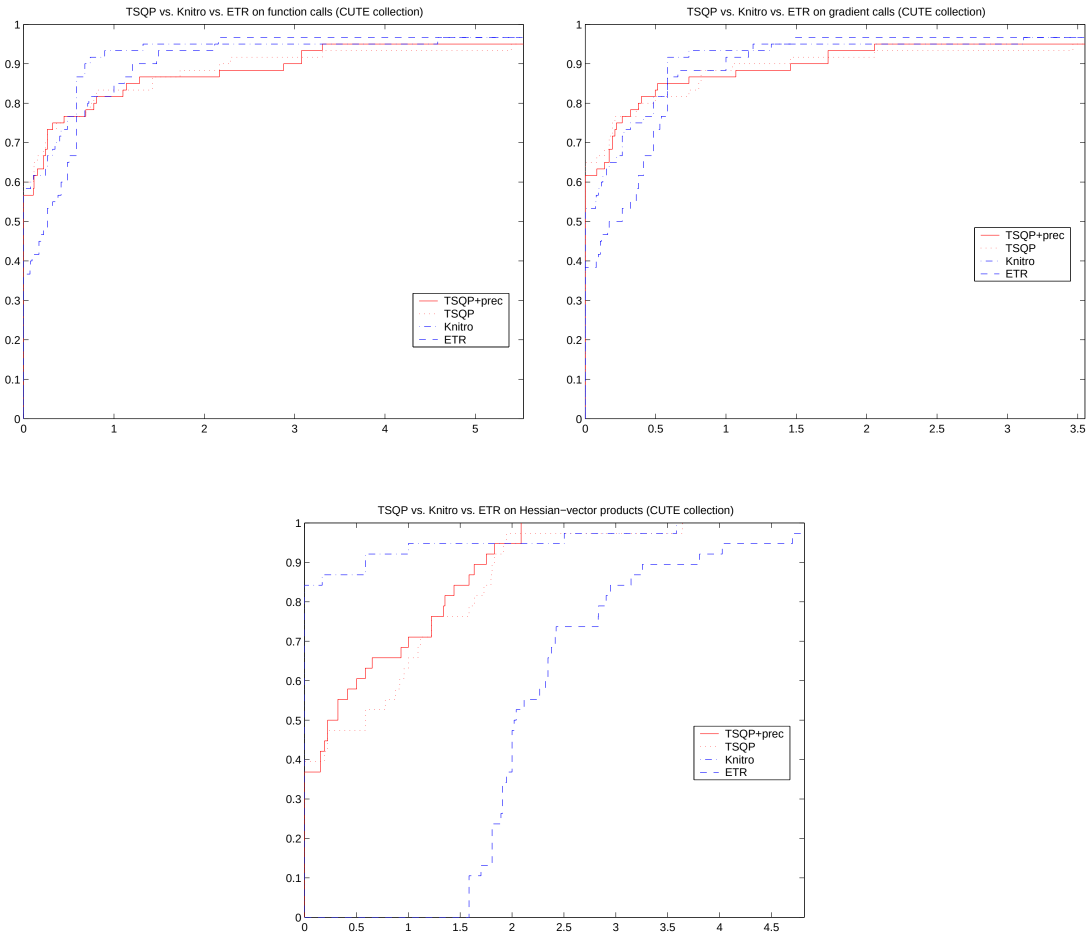

Figure 4.1: Performance profiles of `ETR`, `Knitro`, and `TSQP` on the CUTE problems

Problems	n	m	$n-m$	Problems	n	m	$n-m$	Problems	n	m	$n-m$
aircrfta	8	8	-	hatfldg	25	25	-	hs51	5	3	2
booth	2	2	-	himmelba	2	2	-	hs52	5	3	2
bt1	2	1	1	himmelbc	2	2	-	hs56	7	4	3
bt2	3	1	2	himmelbd	2	2	-	hs77	5	2	3
bt3	5	3	2	himmelbe	3	3	-	hs78	5	3	2
bt4	3	2	1	hs6	2	1	1	hs79	5	3	2
bt5	3	2	1	hs7	2	1	1	hs100lnp	7	2	5
bt6	5	2	3	hs8	2	2	-	hydcar6	29	29	-
bt7	5	3	2	hs9	2	1	1	hydcar20	99	99	-
bt8	5	2	3	hs26	3	1	2	hypcir	2	2	-
bt9	4	2	2	hs27	3	1	2	maratos	2	1	1
bt10	2	2	-	hs28	3	1	2	methanb8	31	31	-
bt11	5	3	2	hs39	4	2	2	methanl8	31	31	-
bt12	5	3	2	hs40	4	3	1	mwright	5	3	2
byrdsphr	3	2	1	hs42	4	2	2	orthregb	27	6	21
coolhans	9	9	-	hs46	5	2	3	powellbs	2	2	-
dixchlng	10	5	5	hs47	5	3	2	powellsq	2	2	-
genhs28	10	8	2	hs48	5	2	3	recipe	3	3	-
gottfr	2	2	-	hs49	5	2	3	trigger	7	7	-
hatfldf	3	3	-	hs50	5	3	2	zangwil3	3	3	-

Table 4.1: Description of the CUTE problems

At first glance, the first two profiles show that the 4 codes are comparable in efficiency: the curves are very close to each others and there is no real winner. A closer look, however, reveals some slight differences. Observe first that `TSQP` has the largest number of wins (values at abscissa 0) with respect to the number of gradient calls (it is equalled by `Knitro` for the number of function calls). On the other hand, `TSQP` has more failures (see the values taken at the largest abscissa): 3 (on `hatfldf`, `himmelbd`, and `powellsq`) instead of 2 for `Knitro` (on `hatfldf` and `himmelbd`) and `TSQP` (on `himmelbd` and `hs56`). In the three cases where `TSQP` fails, $n = m$ (i.e., there is no optimization) and non-convergence arises because the restoration step becomes too large. `TSQP` has also some difficulties on `byrdsphr` (minimization of a linear function on the intersection of two spheres), since at the starting point the Jacobian of the constraints is nearly singular and, again, large restoration steps are generated during the first iterations. For the while `TSQP` has not the possibility to manage such cases with efficiency.

Let us now consider the performance profiles on the number of Hessian-vector products (third picture in figure 4.1). Clearly, `Knitro` is the most efficient code with respect to this criterion. We don't have any clear explanation why there is such an important difference between `Knitro` and its cousin `ETR`. We should also note that `TSQP+prec` and `TSQP` behave approximately the same. To this respect, observe that the dimension $n-m$ of the constraint manifold is usually very small for the selected problems (≤ 3, except on 3 problems for which the value is 5 or 21). Therefore the number of CG iterations is never very large and we believe that this is the reason why `TSQP+prec` cannot benefit from its tangent BFGS preconditioning. We shall see that the situation is quite the opposite for the industrial problems considered in the next section: the number of constraints is small with respect to the number of variables and `TSQP+prec` performs much better that `TSQP`.

To conclude, one can say from figure 4.1 that for these small-size academic problems, `TSQP` is quite competitive with `ETR` and `Knitro`, only slightly less robust. Future research is needed to improve the behavior of the code on problems with singular or almost singular Jacobian matrices.

4.4 Tests on a few industrial applications

In this section, we present numerical experiments with `ETR`, `TSQP`, and `TSQP+prec` on a few real-life test-problems, provided by the lens manufacturing company ESSILOR. The comparison does not include `Knitro`, since for unclarified reasons it was not possible to link this code with a simulator that uses "Matlab engines".

The experiment is limited to 5 nonlinear least-squares problems, taken with their names from the benchmark made up by Jonsson [17]. The problem dimensions are given in table 4.2: n is the number of variables, m is the number of equality constraints, n_r

Problems	n	m	n_r	κ_2
T1	228	4	3194	6.3×10^5
T2	228	4	1850	5.5×10^8
T4	150	4	904	1.0×10^7
T5	150	4	662	3.6×10^7
T6	228	4	3176	2.1×10^9

Table 4.2: Description of the ESSILOR problems

is the number of residuals, and κ_2 is the ℓ_2 condition number of the Hessian of f at the solution. One of the noticeable features of these problems is their ill-conditioning, which varies between 10^5 and 10^9.

The positive thresholds used in the stopping tests are set in the three codes to $\varepsilon^l = \varepsilon^c = 10^{-3}$. None of the codes fails to reached these thresholds in a reasonable amount of iterations.

The performance profiles of the solvers are given in figure 4.2. The most spectacular change with respect to the results obtained on the CUTE collection is the much better performance of `TSQP+prec`. In terms of Hessian-vector products or CPU time, the solver is always the best (its performance profile is the vertical line at abscissa 0, which is hidden by the vertical axis). As shown by the largest abscissa in the last picture, this technique can decrease the computing time by a factor of 2^5. This is essentially due to the fact that for these problems, the computing time is directly proportional to the number of Hessian-vector products (the last two pictures show very similar curves). It is therefore important to limit this number. This is precisely the role of the tangent BFGS preconditioning, which turns out to play a decisive role here.

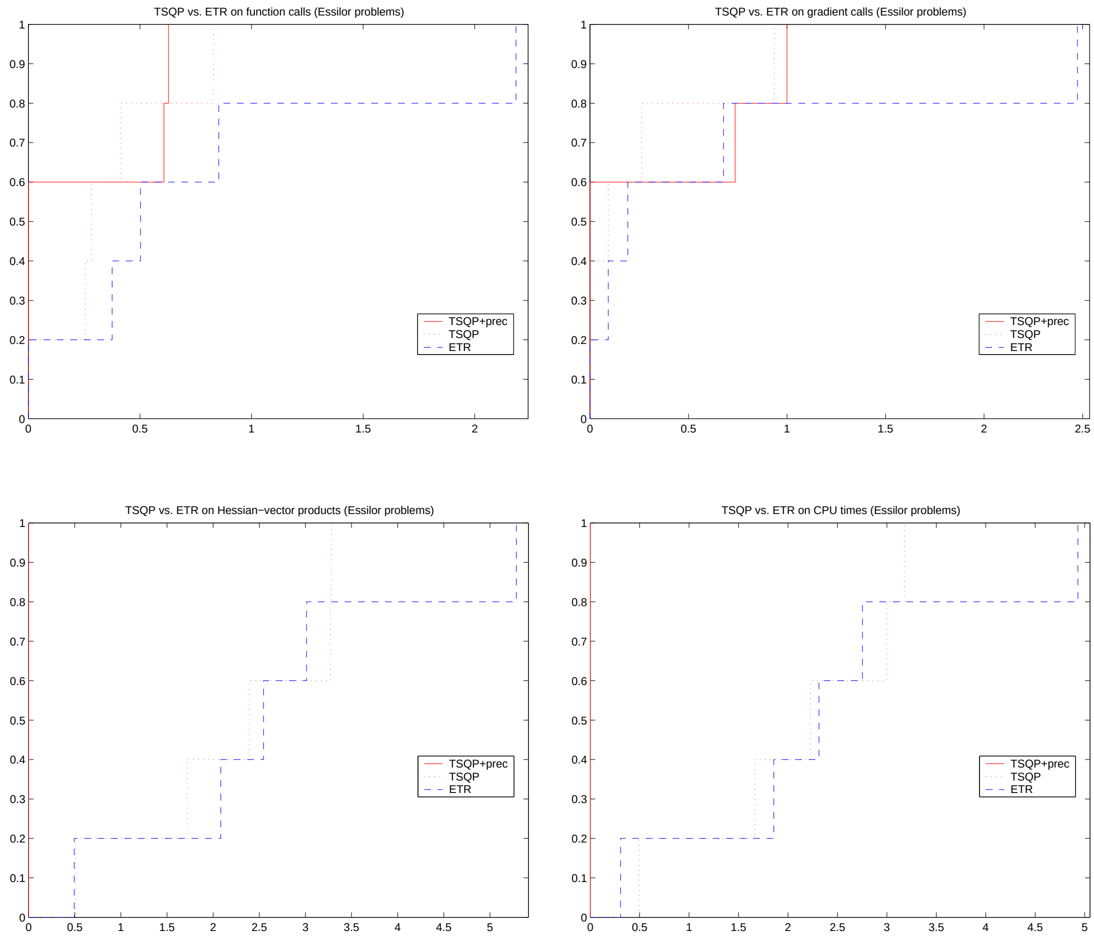

Figure 4.2: Performance profiles of `ETR` and `TSQP` on the ESSILOR problems

5 Conclusion

This paper has presented an elementary truncated SQP approach for solving equality constrained optimization problems. Nonconvexity is detected by conjugate gradient iterations on the linear system formed with the reduced Hessian of the Lagrangian. The convergence of the approach is analyzed. Furthermore, numerical experiment has shown that its efficiency is competitive with the trust region approach, except when the problems present singularity in the Jacobian of the constraints. When the solver uses a tangent BFGS preconditioning, its remarkable efficiency to solve some industrial ill-conditioned problems has been demonstrated.

The algorithm can only find stationary points, since it discards negative or quasi-negative curvature directions. For using these directions efficiently, and therefore being able to find points satisfying the second order conditions of optimality, it would be necessary to study first the correspondence between negative curvature directions for the tangent quadratic problem and negative curvature directions for the merit function Θ_σ, provided one can give a sense to this latter notion when Θ_σ is nondifferentiable.

References

[1] J.T. Betts (2001). *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM.

[2] P.T. Boggs, J.W. Tolle (1995). Sequential quadratic programming. In *Acta Numerica 1995*, pages 1–51. Cambridge University Press.

[3] I. Bongartz, A.R.Conn, N.I.M. Gould, Ph.L. Toint (1995). CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software*, 21, 123–160.

[4] J.F. Bonnans, J.Ch. Gilbert, C. Lemaréchal, C. Sagastizábal (2001). *Numerical Optimization – Theoretical and Practical Aspects*. Springer Verlag, Berlin. (to appear).

[5] R.H. Byrd (1987, May). Robust trust region methods for constrained optimization. Third SIAM Conference on Optimization, Houston, TX.

[6] R.H. Byrd, J.Ch. Gilbert, J. Nocedal (2000). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89, 149–185.

[7] R.H. Byrd, M.E. Hribar, J. Nocedal (1999). An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9, 877–900.

[8] L. Chauvier (2000). *Commande Optimale d'Engins Sous-Marins avec Contraintes.* Thèse de doctorat, Université Paris I (Panthéon-Sorbonne).

[9] L. Chauvier, G. Damy, J.Ch. Gilbert, N. Pichon (1998). Optimal control of a deep-towed vehicle by optimization techniques. In *Proceedings of the IEEE/OES Conference "Oceans'98", Nice, France*, pages 1634–1639.

[10] A.R. Conn, N. Gould, P.L. Toint (2000). *Trust-Region Methods*. MPS/SIAM Series on Optimization. SIAM and MPS.

[11] R.S. Dembo, T. Steihaug (1983). Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26, 190–212.

[12] E.D. Dolan, J.J. Moré (2001). Benchmarking optimization software with performance profiles. Technical Report ANL/MCS-P861-1200, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

[13] R. Fletcher (1987). *Practical Methods of Optimization* (second edition). John Wiley & Sons, Chichester.

[14] A. Fuduli, J.Ch. Gilbert (2002). OPINeL: a truncated Newton interior-point algorithm for nonlinear optimization. Technical report, INRIA, BP 105, 78153 Le Chesnay, France. (to appear).

[15] J.Ch. Gilbert, X. Jonsson (1997). Méthodes à régions de confiance pour l'optimisation surfacique de verres ophtalmiques progressifs. Rapport de fin de contrat Essilor-Inria, référence 1-97-D-577-00-21105-012, INRIA, BP 105, 78153 Le Chesnay, France.

[16] J.Ch. Gilbert, X. Jonsson (2002). BFGS preconditioning of a trust region algorithm for unconstrained optimization. Rapport de recherche, INRIA, BP 105, 78153 Le Chesnay, France. (to appear).

[17] X. Jonsson (2002). *Méthodes de Points Intérieurs et de Régions de Confiance en Optimisation Non Linéaire – Application à la Conception Optimale de Verres Ophtalmiques Progressifs*. Thèse de doctorat, Université Paris VI. (to appear).

[18] M. Lalee, J. Nocedal, T. Plantenga (1998). On the implementation of an algorithm for large-scale equality constrained optimization. *SIAM Journal on Optimization*, 8, 682–706.

[19] N. Maratos (1978). *Exact penalty function algorithms for finite dimensional and control optimization problems*. PhD Thesis, Imperial College, London.

[20] D.Q. Mayne, E. Polak (1982). A superlinearly convergent algorithm for constrained optimization problems. *Mathematical Programming Study*, 16, 45–61.

[21] B. Mohammadi, O. Pironneau (2001). *Applied Shape Optimization for Fluids*. Oxford University Press.

[22] J.L. Morales, J. Nocedal (2000). Automatic preconditioning by limited memory quasi-Newton updating. *SIAM Journal on Optimization*, 10, 1079–1096.

[23] S.G. Nash (1984). Truncated-Newton methods for large-scale function minimization. In H.E. Rauch (editor), *Application of Nonlinear Programming to Optimization and Control*, pages 91–100. Pergamon Press, Oxford.

[24] J. Nocedal, R.A. Waltz (2001). KNITRO 1.00 – User's manual. Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Il 60208, USA.

[25] J. Nocedal, S.J. Wright (1999). *Numerical Optimization.* Springer Series in Operations Research. Springer, New York.

[26] E.O. Omojokun (1991). *Trust region algorithms for optimization with nonlinear equality and inequality constraints.* PhD Thesis, Department of Computer Science, University of Colorado, Boulder, Colorado 80309.

[27] E. Polak (1997). *Optimization – Algorithms and Consistent Approximations.* Applied Mathematical Sciences 124. Springer, Paris.

[28] T. Steihaug (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20, 626–637.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399